A new algorithm for reactive learning of obstacle avoidance

Vicente Ruiz de Angulo

Abstract

Previous work in sensorymotor learning of reactive behaviour with Hierarchical Extended Self-Organizing Maps (HESOM´s)) s examined. A number of observations of the strengths and weaknesses of such a system are outlined and several improvements are suggested. However, real robot learning has requirements that are only fullfiled with a new algorithm presented here. This algorithm is capable of very quick learning due to its dynamic structures. But, more interestingly, it deals more effectivelywith continuous learning anddifferent distributions of data samplings.

Introduction

This report affords the learning of reactive behaviours in autonomous robots. The problem of building a sensor based system under the supervision of a planner [1,2,3,4,5] is taken as example. More concretely we examine the work in [5], in which a Hierarchical Extended Self-Organizing Map (HESOM) [6] learns suitable <*perception, action*> pairs to reach a goal while avoiding obstacles. A perception in their framework is made of a vector of readings of 24 obstacle proximity sensors together with the relative goal direction, and an action is a triple representing an x-translation a y-translation and a rotation represented with respect to the robot current position, represented by a 2 dimensional vector of unitary length.

The HESOM is "extended" because it is trained on the output action in a supervised fashion; it is "hierarchical" because it processes the input perception in sequence. First, the readings of proximity sensors are processed by a supernet SOM, and second the goal direction is processed by a SOM subnet associated to the winning neuron in the supernet.

The simulations in [5] generate data by creating an environement with obstacles and fixing a goal point in the space, and afterward generating a random initial point from which the planner begins to teach the network which are the more appropriate actions. When a local minima or the goal are reached, another initial point is generated. In this way a large number of examples for the network are recorded and they are iteratively presented in "batch mode". The same set of examples is used to evaluate the system averaging the quadratic errors between the network outputs and the planner outputs.

Although the purpose of the paper is the presentation of a completely new architecture and algorithm, it is interesting to note some of the weakenesses of the system, and point out how they could be solved.

Modifications to the HESOM system

These are the changessuggested for the system examined in the previous section:

Performance measures.
First of all, the system needs a method to compare performances more in accordance with the desired behavior than merely the quadratic error between the network and the planner movements. What really counts is whether the robot collides or not, while the exact coincidence of the direction chosen by the network and the planner is less relevant. The exact coincidence is even not very appropriated because the network must profit from the fact that, unlike the planner, its movements are not discretized.
The suggested measure is the number of collisions made by the network in the path going from some previoulsy fixed initial points to a certain goal. (better not to use exactly the same paths for testing and learning, if the last ones are fixed). This new measure can be used jointly with the other.

Goal angle representation.
The representation of the angles in two dimensions can be shown to be flawed, exhibiting a distortion of the real distances and other strange side-effects. An appropriate representation with a convenient distance measure expressing the neighborhood relations of the angle space is easily found. This distance should be used both for determining the nearest cell and must

regulate also the neurons movements with its gradient. In addition a subnetwork of circular topology can be used.

Neighborhoods.
A different treatment should be given to the neighborhood scheduling at the different levels of the network (in the original description of the HSOM [6] all the levels in the architecture share the same learning schedules). At the clustered perception level, the neighborhoods can produce destructive interferences, so a scheduling reducing early the neighborhood to zero is advised. On the other hand, at the goal direction level, the interference is almost always constructive. Thus a flatter and higher scheduling seems better.

Sample generation.
If temporal distribution of the samples is equally distributed, Kohonen maps can be used for on-line learning without major modifications.
Next, and as a consequence, a method of random generation of samples a little bit nearer to real robot behavior is suggested. The main points of this modification are:

a) Random generation of the goal and the initial point for the robot. Each time the robot reaches the goal or finds a local minima, a new goal and initial point are generated.

b) Unless a collision is made, the movement really performed by the robot during the learning is the output of the network. The rationale for this is, that the network must see and improve its behaviour in the situations to where itself uses to arrive, instead of those situations tipically produced by the planner.

The problem of the sample distribution

The above suggested sample generation procedure was tested, and immediately a new problem made itself evident. The cause came from point a) in the generation procedure: In a complex environment, the robot very often falls in a local minimum after a few steps. Thus during the learning, the robot is almost always in “free space” or sensing obstacles that do not put it into difficulties. Therefore, in the distribution of samples the easiest situations are majoritary and the net devotes a great part of its cells to these zones, neglecting the difficult ones.

This is a serious problem and these are some possible solutions for it:

- To use initial points of the robot carefully chosen to avoid minima. It was like that in the original system, but returning to this kind of strategy is unattractive, because it is difficult and embarrasing to apply with a real robot.
- To use a very crowded environment with a great density of obstacles. Two drawbacks: the space would be full of local minima, and the success of the learning depends on the environment. If by any reason, a real (or not) robot spend some time in the free space during the learning, much "difficult" knowledge will be erased.

-Error driven learning. Theoretically appears as the right solution. But in practice is extremely slow and low performing.

Before introducing some new solutions, I would like to make an observation. It is my opinion that the problem can actually be divided into two problems. Suppose that to learn the task correctly, any learning algorithm needs at least a certain number of interesting samples. Suppose also that you have a sample generation procedure producing interesting samples with a frequency that is half of the produced by a second one. Then, the minimum number of total samples needed to learn the task is double with the first procedure . On the other hand, a complete different problem is whether we can learn or not correctly the task with the first procedure, even with a huge number of examples, because the algorithm anyway needs the correct proportion of samples to learn. Therefore the two problems are:

1) Generating enough frequency of interesting samples.
2) Given a number enough of samples, being able to learn the task, indepently of the sample distribution. We need to solve this problem because in real life we cannot control that the distribution is always good enough

For the first problem a solution can be the same that the one proposed in the previous section, but now, only the goal is generated randomly when it is reached or the robot finds a local minimum. Like this, the resemblence with real robot learning is even stronger than before (a robot cannot move randomly and instantaneoustly to other parts of the environment), and in all the cases of local minimum, the robot continues in a point that must be very close to some object. This can be combined with a greater density of objects.

To overcome the second problem, a new algorithm has been devised, which allows also quick learning. Around a basic framework, very different versions can be developed depending on

some elections, such as using one or two level architectures and using continuous or discret units.

Basic arquitecture

The one-level version for discrete units bears some resemblence with some aspects of the GAL algorithm [7]. The main difference is that GAL is conceived for categorization, while the proposed algorithm aproximates real functions. There exists the possibility of using GAL with real functions (as explained by the author) discretizing the ouput range, and taking each discretized value as a different class, but this solution is highly impractical, specially for multidimensional outputs. I think a much proper approach is the one presented here.

The basic architecture is a constructive Winner-Take-All network with an output vector and a counter associated to each cell. The cell i is denoted with a triple (W_i, n_i, O_i), representing the weight vector, the counter and the output vector. This is the proposed algorithm:

For each pattern sample (X_p, D_p)

Find cell l that minimizes distance(X_p, W_l)
if too-much-error then New cell j= $(X_p, 1, D_p)$
else $n_l = n_l + 1$

$$O_l = \frac{((n_l - 1)\, O_l + D_P)}{n_l}$$

As you can see, it is extremely simple. There are no neighborhood schedulings (no neighborhoods at all), nor learning or modification in the cell weights, nor computations of implicit dimenstionality,...nor learning parameters (neither a learning rate!).
The reasons for expecting quick learning, is that every part of the input space that reveals as being bad mapped by the network, is inmediately corrected by the creation of a new unit. Each unit has exactly the mean of all the patterns for which it was the winner and gave an approximately good output. As the centers (W_i) of the cell do not move, the output values learned by a cell at the begining must remain perfectly valid (as a matter of fact, not completely because blablabla..) also at the end of the learning. This is another reason that helps fast learning, so as the fact that the output of the network is always well-defined in all the output space (unlike architectures similar to Fritzke's self-growing cells [8], which produces a zero value in a big part of the input space)

For the application of this algorithm to our problem the *too-much-error* condition can take the form:

too-much-translation-error or *too-much -Turning-error* or *collision*,

and *too-much-translation-error* could be

$||dmov - mov|| >= k_1$

being |dmov the vector $\Delta x, \Delta y$, the correct traslation movement given by the plannerand mov the one produced by the winning cell |. k_1 must be not too little because, the ideal output for a device able to produce any real output, like the network, is not the same that the optimal movement for the planner, that only consider some few discrete values.
Thus a natural value for k1 is the discretization interval used by the planner in both axes.

For k_2, instead, given that the planner generates three possible turning angles , it seems better to divide the total range of possible angles in three equally sized intervals, and *too-much-turning* angle would be determined by

if (*desired-angle*= 0) then *too-much-turning-error* = (| *ang* | >= 1/6 *range*)
else *too-much-turning-error* = (| *ang* - *dang* | >= 1/3 *range*)

where *dang* is the correct turning angle given by the planner, ang is the one produced by network and, range, the difference between the maximum and the minimum turning angles tested by the network.

Limiting resources in a continuous learning setting

If the mapping to be learned is not deterministic, i.e., if there can be different correct outputs for the same inputs in different moments, an algorithm of this kind will tend to add an infinite number of neurons to correct totally the error, which is imposible, becase the minimimum reachable error in a certain point is the variance of the mapping in that point. *I think* that this kind of situation is improbable in our simulated robot and environment, but would be surely frequent in a real robot with few and noisy sensors. To avoid this, another condition must be added in the "if" part of the algorithm, not allowing the creation of the new cell if $distance(p_p, W_l) < cte$, where cte is the minimum distance allowed between two cells.

If you don't like this constant, think that GAL or Fritzke´s algorithm, and indeed, any constructive algorithm tends to add infinite units in a continuous learning, noisy-data aplication (also if the application is noisy-free and not changing. Only an infinite number of sample or continuous learning is needed to see this tendency).
The resolution constant regulates the maximum resolution reachable by the network. This constant is related with too-much-error, and one can anullate the effect of the other. For example, when there are lots of noise, too-much-error will be parctically always true, and only the resolution constant will take the charge of limiting the number of units.
The ideal value for the resolution constant in a concrete application is in general not clear. But this is the universal problem of overfitting, that has to be solved with any kind of estimator. The number of hidden neurons in a back-prop network or the dimensionality of the extended Kohonen maps, play a role similar to the one of the resolution constant here.
After maximum resolution is reached, some fine tuning is possible depending on the kind of noise. This tuning would be not the same that the one suggested by Alpadyn, which refers to a categorization application, and with a fixed learning set. This question can be treated more in depth when really dealing with an application with noise.

Changing environement

The formula

$$O_l=\frac{(n_l - 1)\, O_l + D_P}{n_l}$$

is a particular case of the rule:

$$O_l= \beta\, O_l + (1- \beta)\, D_p.$$

With this kind of updating the current output produced by the cell is always a weighted mean of past data:

$$O_l = c_1\, D(1) + \ ... + c_m\, D(m)$$

Where m is number of times that cell has won up to now,D(i) was the desired output when cell l won the time i., and $c_1 + \ ... + c_m$ is always 1. As was said before, the particular form proposed before produces always the exact mean of all the data seen, and thus $c_i= 1/m$.

This way of updating is not very adaptive when the system is old. For example, if the environment changes when the cell l has won 1000 times, to be half-adpated, that is, to be taken into account with the same strength than the older, cell l has to win 1000 more times.
But we can use other β that make the weighing coefficients of the old and new patterns different according to their age. One of these is a β = constant. With this updating the coefficients become:

$c_i = (1-\beta)\,\beta^{m-1}$ if i>1
$c_i = \beta^{m}$ if i =1

It can be seen that in the long term the coefficients decrease exponentially in time and so the responsability of the data far in time is negligible. The lower is β, the higher is the adaptability to the more recent data. Of course, there exists the plasticity-stability dilemma, as often mentioned by Grossberg [10], but for us β=.2 seems reasonable .

However, there is a problem with this kind of β. At the begining of the learning there is a contraituitive effect, taking the older pattern most of the reponsability for the output. Taking for instance β= .2 as before, and m=3, the coefficients are:c_1= .64, c_2= .16, c_3= .2.

A more sophisticated β that avoids this effect is

$$\beta = \frac{\sqrt{n_l}}{1+\sqrt{n_l}} \quad \text{and} \quad (1-\beta) = \frac{1}{1+\sqrt{n_l}}$$

The coefficients in this case are:

$$c_i = \frac{\prod_{h=0}^{m-i-1} \sqrt{m-h}}{\prod_{h=m}^{i} \sqrt{h}+1}, \qquad \text{if } i \neq 1$$

$$c_1 = \frac{\prod_{h=m}^{2} \sqrt{h}}{\prod_{h=m}^{2} \sqrt{h}+1}$$

I advice to run a little program producing the list of all the coefficients given m. for personal experimentation. Perhaps multiplying all the coefficients by $(1+\sqrt{n_l})$ in such a way that the higher coefficient becomes 1, it is easier to check the following properties:

- The coefficients grow form past to future.
- For little m, there are not contraituitive effects, i.e., the former condition still holds.
- The number of recent data that counts to determine the outputs grows with m. The same happens with the number of the more remote data that does not count, but this one grows more quickly.

This is the updating rule I suggest for general use, even when the environment is not changing.

All the above can be combined with reseting n_l to some low number, periodically or when an important environmental change is known to be taking place.

Removing cells

Another refinement of the algorithm can be made removing unuseful cells. In the course of the learning, the wining field of some cells can be made very small because of the creation of many cells in the sorroundings or anyway if by some reason some cells win rarely. It is not difficult to imagine some strategies less or more sophisticated (among the last ones looking the second winning cell) to hold these situations, and I will not insist more upon this point here.

Most of the constructive algorithms like GAL or Fritzke´s Growing Cells have also mechanisms to remove superfluous cells. We adopt a method that is a hybrid of the one used in the two mentioned algorithms.
The problem in GAL is that there are sudden error increases when a cell is removed that must be recovered slowly. This is surely due to the fact that the removing decision is based in only one pattern. Another inconvenient is that there is a sleepping phase which cannot be used for learning. The last difficulty, that should not be minimized, although it is not mentioned, is the generation of random patterns (think for example in our application, how to generate patterns that make sense as robot perceptions).
I suggest that the decision of removing must be based in the error of the second best cell, but using training patterns instead of random ones, and a counter to take into account several patterns instead of one. We call p_i this counter for the cell i. A cell is now a quadruple (W_i, n_i, O_i, p_i), and the algorithm:

For each pattern sample (X_p, W_l)

Find the best cell and second best cell that minimize distance(X_p, W_l).
Call them ffand s respectively
if *too-much-error*(O_f and dist(X_p, W_f) < cte then New cell j= (Xp, 1, Dp, 0)

else $n_f = n_f + 1$

$$O_f = \frac{\sqrt{n_f} O_f + D_p}{\sqrt{n_f} + 1}$$

when not *too-much-error*(O_s) then $p_f = p_f + 1$
when (n_f *mod* ctea) = 0 then
if p_f> cteb then remove cell f
else $p_f = 0$

ctea can be set to 8 and cteb to 6 for instance in our application. I think that these constants can produce good results in a wide range of reasonable values, and therefore we have not to worry too much about them. If you don't like either of these constants, think that Frittzke's algorithm and GAL both use also (implicitly) two constants only to remove cells.
This is a way of using the counter p_l, but there are others. For example, using it to have always the number of consecutive times that a second best cell has won, and removing the cell when this counter is greater than a certain number.
Or never resetting it, adding 1 when second cell is correct, and multiplying always by a *decay* constant, such that 0< *decay* <1. The cell l is removed when p_l is greater than a *top* fixed number. In this way, more recent failures of the second cells are more important than old ones, and there are not preespecified moments to decide to remove the cell. Unfortunatelly,one has to take care of the relationship between *decay* and *top*, because if for instance *top* is very big and decay is very little, top can not ever being reached. For this reason, in a first stage, I think it is better to try with the explicited algorithm, because the parameters are intuitive, and as I said the results sould be not very sensitive to their value.

Goal direction is special

Our application has a particular feature: one of the components of the inputs, the goal direction, is very important for determining the output. If all sensor values are equal,

variations in the goal angle change completely the correct output. No other input component is so determinant.

Note that the basic architecture has only a level, not two, as the original system, and in principle the goal direction is treated exactly in the same way that any sensor. This will not avoid the basic architecture to work well, but taking into account the particularities of the goal direction can make the system much more compact, needing less cells to learn the task. There are several means to hold with the difference, and I will mention two of them here.

The more simple one is using hyperellipsoids instead of hyperspheres to measure distances. I only mean with this that instead of using
$||Xp - W||^2$ we could calculate $\sum_i a_i (w_i - x_i)^2$. In our case, we could take all the $a_i = 1$ except the corresponding to the angle that should take a higher value. It is worthy to note that the same effect of introducing the c_i in the distance, can be obtained without them by only scaling appropriately the input components (scaling by x component i means the multiplication of a_i by x^2). Moreover, if the variance of the input components in the input distribution are different, implicitly we are given different importance to the components. As the angle direction is of a nature different than the one of the sensors, it would probably have by itself a different variance. The simple fact of changing the representation of the angle can modify this aspect of the learning. Then, before to assign a_i's, the variance of the components must be examined in the representation used by the net. Whatever the method employed, it is suggested that making the angle component between three and ten times more important than the others, the most compact representations would be obtained,

The other solution proposed is going directely to a two level architecture as in the original sysem, with one level for sensors, and another for the goal directions. The advantage of this kind of architecture is that, although the number of total neurons can be high, most of them are in the higher goal direction level with very few parameters. Also, the hierarchical structure allows more velocity in the search of the best-matching neurons.
But it has a big disadvantage, specially for constructive algorithms: It is difficult to assign the responsability of errors to one or other level. This repercutes in much lower velocity. It is an instance of the (I think) universal trade-off between compactness of the representation and ease and velocity of learning. Shall we try to minimize this drawback.
I will not specify an algorithm because there are lot of possibilities. Everything , including refinements of the algorithm like limiting resolution or removing cells must or can be duplicated, which in some cases can reveal suprefluous. Nevertheless I will make some suggestions:

- It is possible to use initial standard goal subnets when a new perception cell is created. With standard subnets I mean that they have a standard number of goal cells (10 for instance), regularly placed in the angle range (but don't make coincide a cell with angle 0), and with an initial standard output that is set to that ideal in free space.However when the subnet is created, the nearest standard cell to the data angle must be removed and substituted by one placed in the data angle and with output, the desired correct output. After the creation, everytime a standard angle cell fails by first time (i.e., if that angle cell is still standard), the output must not be modified with the standard updating rule; it is better to set it to the correct output. The decision to create a new perception cell can be based on how much has been failing the winning angle cell. A simple criterium would be to create the new perception cell if the winning angle cell has failed another time, i.e., if it is no more standard.

- An alternative to the former strategy is to make the new created perception cell to inherit the same angle subnet that its father (the perception cell whose failure causes the creation). But, at the creation moment, the same removing and substituting operation mentioned above of the nearest inherited angle cell must be performed.
This alternative is fundamental if addition of new cells in the angle level is considered. At least if the strategy outlined in the next point is not followed

- Other strategy, very recomended if addition of angle cells is considered and the last suggestion is not followed (i.e. , if the new angle subnets must learn from the scratch,), is to limit the receptive fields of the angle cells. If data angle is at a distance to the winning cell greater than 60 degrees, for instance (it is really not very important the exact value), the network output must be given by formula that must produce the ideal movements in free space, and not by the output of the winning cell. Of course when this formula fails, a new cell must be created.
If we assume growing angle subnets, I think it is better to limit this growing imposing a top number of cells in a subnet (10) instead of limiting the resolution. The impossilbilty of creating a new angle cell, would cause the creation of a new perception cell.

- In the angle level it could be advisable to move the winning cell position but with a low learning rate. This could be done towards the data position or instead, in a data driven error way, taking into account the failure or not failure of the winning cell and perhaps more things.

-Cells in the angle level could also be removed, but I recommend to be very exigent before removing one (for example ctea =5 and cteb=4). Instead, removing cells at the perception level must be easier, but more patterns must be taken into account (ctea= 15 cteb= 10).

Conclusions

The HESOM network designed in [5] appears appealing as complement for a discretized planner. First, the HSOM is quicker, and second, produces smoother paths. However it does not fullfil all the conditions that must be met to be efficient in a real robot. A number of the collisions that the system produces after learning could still be predicted from the sensor map and avoided. The training must be also accelerated to be useful. A different goal angle representation and the division of the neighborhood parameters of the supernetwork and the subnetworks help to mitigate these problems.

But, to really overcome them, a new type of architecture is required. In designing the new system, other desirable properties were taken into account: on-line and continuous learning, and robustness against the order and frequency of presentation of the different example types. In doing this, we created dynamics structures and learning rules that explicitly express the stability-plasticity dilemma. The equilibrium between these two poles can be be convenientely regulated. Initially there is a great plasticity and learnig is almost instantaneous. In the long term, the old information is modified at a slower pace. But if something unexpected happens, the equilibrium can recover its former plasticity.

References

[1] Millán, J del R. (1995): "Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot", Robotics and Autonomous Systems, 15(13), pp. 275-299.

[2] Heikkonen, J. Millan, J. del R., Cuesta, E. (1995): "Incremental learning from basic reflexes in an autonomous mobile robot", International Conference on Engineering Applications of Neural Networks (EANN95), pp.119-126.

[3] Heikkonen, J. Koikkalainen, P., Oja, E. (1993):." behaviour learning by self-organization", International Conference on Artificial Neural Networks (ICANN93), pp. 262-267.

[4] Knobbe, A.J., Kok, J.N., Overmars, M.H. (1995): "Robot motion planning in unknown environements using neural networks", Internatinal Conference on Artificial Neural Networks (ICANN95), pp. 375-380

[5] Versino, C. and Gambardella, L.M.(1996): "Learning fine motion by using the hierarchical Extended Kohonen Map", Internatinal Conference on Artificial Neural Networks (ICANN96), pp. 221-226, Springer Verlag.

[6] Ritter, H., Martinetz, T., Schulten, K. (1992): "Neural Computation and Self-Organizing Maps. An introduction", Addison-Wesley Publishing Company.

[7] Alpaydin (1991): "GAL: networks that grow when they learn and shrink when they forget", TR-91-032, International Computer Science Institute.

[8] Fritzke, B. (1993): "Growing Cell structures. A self-organaizing Net for unsupervised and supervised learning", TR-93-026, International Computer Science Institute.

[9] Carpenter, G.A. y Grossberg, S. (1988): "The ART of adaptive pattern recognition by a self-organizing neural network", Computer, Vol. 21, pags. 77-88.